Efficient Proofs of Retrievability using Expander Codes

Françoise Levy-dit-Vehel[1] and Maxime Roméas[2]

[1] LIX, ENSTA Paris, INRIA, Institut Polytechnique de Paris, 91120 Palaiseau, France
levy@ensta.fr
[2] LIX, École polytechnique, INRIA, Institut Polytechnique de Paris, 91120 Palaiseau, France
romeas@lix.polytechnique.fr

Abstract. Proofs of Retrievability (PoR) protocols ensure that a client can fully retrieve a large outsourced file from an untrusted server. Good PoRs should have low communication complexity, small storage overhead and clear security guarantees. We design a good PoR based on a family of graph codes called expander codes. We use expander codes based on graphs derived from point-line incidence relations of finite affine planes. Høholdt *et al.* showed that, when using Reed-Solomon codes as inner codes, these codes have good dimension and minimum distance over a relatively small alphabet. Moreover, expander codes possess very efficient unique decoding algorithms. We take advantage of these results to design a PoR scheme that extracts the outsourced file in quasi-linear time and features better concrete parameters than state-of-the-art schemes w.r.t storage overhead and size of the outsourced file. Using the Constructive Cryptography framework of Maurer, we get sharper and more rigourous security guarantees for our scheme than the ones given by the usual ϵ-adversary model. We follow an unbounded-use audit procedure to ensure that the extraction of the outsourced file will succeed w.h.p.. The properties of our expander codes yield an audit with communication complexity comparable to other code-based PoRs.

Keywords: Proofs of Retrievability · Expander Codes · Outsourced Storage

1 Introduction

1.1 Context and state-of-the-art

With the continuous increase in data creation, individuals and business entities call upon remote storage providers to outsource their data. This new dependency raises some issues, as the storage provider can try to read and/or modify the client's data. Besides, when a client does not often access his data, the service provider can delete it to make room for another client's data. In this context, it appears important to deploy client side protections designed to bring security

guarantees like confidentiality and integrity. In this work, we focus on the following problem : given a client who stored a file on a server and erased its local copy, how can he check if he is able to retrieve his file from the server in full ? Addressing this issue is the goal of a class of cryptographic protocols called Proofs of Retrievability (PoRs).

The first PoR scheme was proposed in 2007 by Juels and Kaliski [10] and was based on checking the integrity of some sentinel symbols secretly placed by the client before uploading its file. This scheme has low communication but its drawback is that it is bounded-use only, as the number of possible verifications depends on the number of sentinels. Shacham and Waters [17] proposed to correct this drawback by appending some authenticator symbols to the file. Verification consists in checking random linear combinations of file symbols and authenticators. Then comes a few PoR schemes based on codes. Bowers *et al.* [3] proposed a double-layer encoding with the use of an inner code to recover information symbols and an outer code to correct the remaining erasures. Dodis *et al.* [4] formalize the verification process as a request to a code which models the space of possible answers to a challenge. They use Reed-Solomon codes to design their PoR scheme. In 2013, Paterson [15] laid the foundation for studying PoR schemes using a coding theoretic framework. Following these ideas, Lavauzelle and Levy-dit-Vehel [11] (2016) used the local structure of the lifted codes introduced by Guo *et al.* [5] to build a PoR scheme, that compares favourably to those presented above w.r.t. storage overhead. In 2022, Levy-dit-Vehel and Roméas [12] proposed a framework for the design of secure and efficient PoR schemes based on Locally Correctable Codes. They also reevaluated the security and the parameters of the [11] PoR scheme.

In this work, we design a PoR scheme based on expander codes. In 1996, Sipser and Spielman [18] introduced these codes that are based on expander graphs. Expander codes are linear codes obtained by taking a d-regular expander graph G and a linear inner code $\mathcal{C}_0 \subseteq \mathbb{F}^d$. The codewords are labelings of the edges of the graph satisfying the following condition : for each vertex v of G, the labels on the d edges incident to v form a codeword in $\mathcal{C}_0$. Expander codes possess very efficient unique decoding algorithms. We will use an erasure decoding algorithm derived from [18,21] during the extraction phase of our PoR.

The expander codes used for our PoR scheme are based on a family of graphs with excellent expansion. These graphs are derived from point-line incidence relations in the affine plane $\mathbb{F}_q^2$. The expansion of these graphs was studied by Tanner in 1984 [20]. A line of work of Høholdt *et al.* [6,7,2] studied the dimension and minimum distance of expander codes based on the previously mentioned graphs with Reed-Solomon codes as inner code.

Finally, we use an audit procedure for generic erasure codes adapted from [10,17] in the Constructive Cryptography (CC) framework of [13] by Badertscher and Maurer [1]. We also prove the security of our PoR scheme using the CC security model for PoRs of [1].

1.2 Contributions

We use an audit procedure from [10,17] translated in the CC framework by Badertscher and Maurer [1] to design a PoR scheme based on expander codes. Recall that an expander code is constructed using a regular expander graph, a so-called inner linear code and, for every vertex v of the graph, an ordering on the edges incident to v. A codeword is a labeling of the edges such that, for every vertex v of the graph, the vector supported by the edges incident to v is a codeword of the inner code.

By encoding the client's file with a well-chosen expander code, we manage to design a PoR scheme with asymptotically :

1. Storage overhead linear in the size of the outsourced file $|F|$.
2. Communication complexity in $\mathcal{O}(|F|^{1/3} \cdot \log |F| \cdot \sigma)$ for σ bits of statistical security.

Furthermore, we give concrete parameters for file sizes ranging from a few Mo to hundreds of Go. Our parameters are better than the ones of [11] for usual lifting parameters such as $m = 2, 3$. When using the same alphabet size and security parameters as other code-based PoRs [10,17,11], our scheme is capable of reaching higher rates and storing larger files. We give parameters and comparisons with other PoRs in sec. 3.4.

We optimize the parameters of our scheme using expander codes based on q-regular graphs derived from point-line incidence relations in finite geometries. The properties of these graphs and of expander codes based on these graphs are studied in [20,6,7,2]. We chose these graphs because they have very good expansion and a good ratio between their regularity q and their number of edges q^3. We show that these two facts combined permit us to reach lower communication complexity and storage overhead than previous code-based PoRs.

Moreover, these graphs exist for every prime power q. By choosing q to be a power of 2 and a Reed-Solomon code of length q as inner code, we can use the erasure decoder for Reed-Solomon codes of [19] with complexity $\mathcal{O}(q \log^2 q)$. Using this decoder along with a fast unique erasure decoding algorithm for expander codes [18,21], we are able to extract the outsourced file in quasi-linear time $\mathcal{O}(q^3 \log^2 q)$ in the input size $Rq^3 \log q$, where $0 < R < 1$ is the rate of the code.

Organization of the paper. In sec. 2, we give the required material to understand our PoR construction. Then, in sec. 3, we describe our audit procedure and we optimize our PoR by choosing a good graph, a good inner code and by tuning its parameters. Finally, we compare the performance of our PoR against other schemes in sec. 3.4.

2 Background

2.1 The Constructive Cryptography model

The CC model, introduced by Maurer [14] in 2011, aims at asserting the real security of cryptographic primitives. To do so, it redefines them in terms of so-

called *resources* and *converters*. In this model, starting from a basic resource (e.g. communication channel, shared key, memory server...), a converter (a cryptographic protocol) aims at constructing an enhanced resource, *i.e.*, one with better security guarantees. The starting resource, lacking the desired security guarantees, is often called the *real* resource and the obtained one is often called the *ideal* resource, since it does not exist as is in the real world. An example of such an ideal resource is a confidential server, where the data stored by a client is readable by this client only. The only information that leaks to other parties is its length. This resource does not exist, but it can be emulated by an insecure server on which the client uses an encryption protocol where the encryption scheme is $\mathsf{IND}-\mathsf{CPA}$ secure. We say that this *construction* of the confidential server is secure if the real world - namely, the insecure server together with the protocol - is *just as good* as the ideal world - namely, the confidential server. This means that, whatever the adversary can do in the real world, it could as well do in the ideal world. We use the fact that the ideal world is by definition secure and contraposition to conclude. This construction notion is illustrated in fig. 2.

The CC model follows a top-down approach, allowing to get rid of useless hypotheses made in other models. A particularity of this model is its composability, in the sense that a protocol obtained by composition of a number of secure constructions is itself secure.

We give the required material to understand how we use CC in app. A. We follow the presentation of [9].

Server-Memory Resources We recall the constructions of [1,12] that we will use in this work. The first resource is the basic server-memory resource (SMR) denoted by $\mathbf{SMR}_{\Sigma,n}$ where Σ is the alphabet and n the memory size. The resource allows the client to read and write data blocks that are encoded as elements of a finite alphabet Σ via its interface C. The interface C_0 is the initialization interface used to set up the initial state of the resource. The server can be "honest but curious" by obtaining the entire history of accesses made by the clients (a log file) and reading their data at interface S_H. The server can also be intrusive and overwrite data using its interface S_I when the resource is set into a special write mode. This write mode can be toggled by the distinguisher at the world interface W. The specification of the resource $\mathbf{SMR}_{\Sigma,n}$ is given in fig. 1.

We recall the figure 2 given in [1] to illustrate the CC construction notion on SMRs. The SMR security guarantees can be augmented to provide authenticity by using a suitable protocol in this construction notion. This new server-memory resource, called authentic SMR and denoted by $\mathbf{aSMR}_{\Sigma,n}$, is introduced in [1]. In the **aSMR**, the behavior of the server at its interface S_I is weakened as the server cannot modify the content of data blocks but is limited to either delete or restore previously deleted data blocks at this interface. A deleted data block is indicated by the special symbol ϵ. Thus, if we store a codeword on the **aSMR**, the adversary can only introduce erasures and not errors.

We use the $\mathbf{aSMR}_{\Sigma,n}$ specification of [12] because it is tailored for code-based PoRs with its $\mathcal{O}(\log n)$ communication complexity per read query. Indeed,

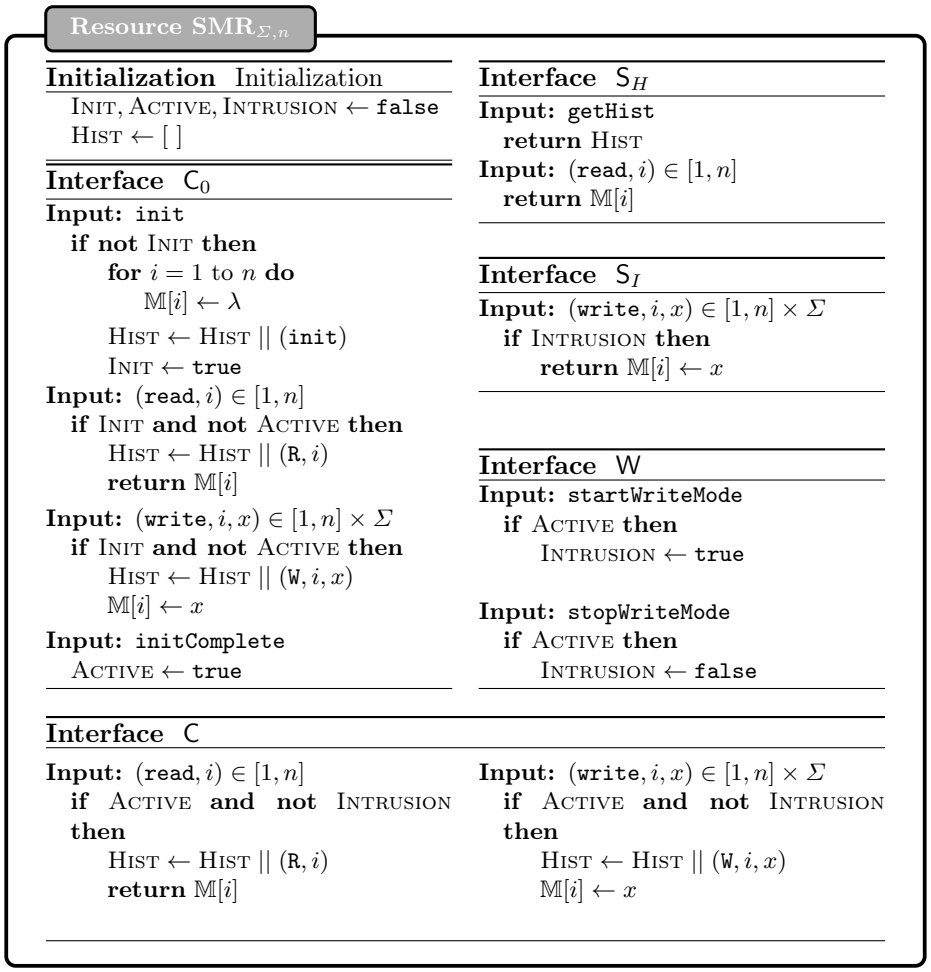

Fig. 1. Description of the basic server-memory resource of [1].

code-based PoRs require a large number of read queries and only one write query to outsource the encoded file. The **aSMR** resource is described in fig. 3 and is constructed in [12] using a simple MAC-based protocol. Each symbol is stored alongside a MAC tag, this yields a storage overhead of κn where κ is the length of a MAC tag.

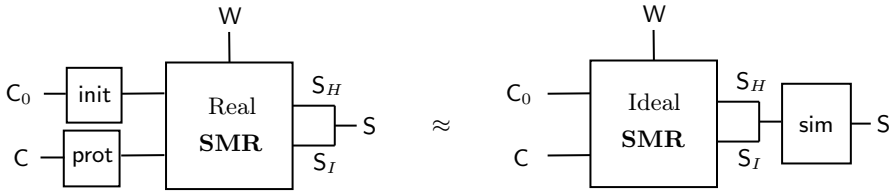

Fig. 2. Illustration of the construction notion for SMRs. On the left, we have a real **SMR** with a protocol for the client. On the right, we have an ideal **SMR** with stronger security guarantees. The construction is secure if there exists a simulator that makes these two resources (computationally or statistically) indistinguishable.

2.2 Proofs of Retrievability

Proofs of Retrievability (PoR) are cryptographic protocols whose goal is to guarantee that a file stored by a client on a server remains retrievable in full. PoRs thus involve two parties : a client who owns a file F and a server, here modelled as an SMR, on which F is stored.

We use the CC based definition of PoR security as presented in [1]. Namely, a PoR scheme is composed of a pair of converters $\mathsf{por} := (\mathsf{por\_init}, \mathsf{por\_audit})$ and works in three phases :

- *An initialization phase.* The client converter init encodes the file F into $\mathbf{Init}(F) = (\tilde{F}, \mathtt{data})$. The converter sends `data` (*e.g.* keys, etc.) to the client, then it sends $\tilde{F}$ to the SMR with a `write` query and erases F from the client's memory.
- *An audit phase.* The client converter audit probes some symbols of the server's memory and outputs `accept` if it believes that the file is retrievable in full and `reject` otherwise.
- *An extraction phase.* If the client has been convinced by the audit phase, he can send `read` to recover his whole file with high probability.

A PoR scheme is considered secure if it constructs an ideal abstraction of PoRs (introduced in [1]). This abstraction consists of an ideal SMR that sees the client's interface augmented with an `audit` mechanism. On an `audit` request, the resource checks whether the current memory content is indeed the newest version that the client wrote to the storage. If a single data block has changed, the ideal audit will detect this and output `reject` to the client. In case of a

Resource aSMR$_{\Sigma,n}$

The **aSMR** definition is identical to **SMR** except for the influence of an adversary at interface S_I and the addition of a version number ctr.

Interface C

Input: $(\texttt{read}, i) \in [1, n]$
  if ACTIVE **and not** INTRUSION **then**
    HIST $\leftarrow$ HIST $||$ $(\texttt{R}, i)$
    return $\mathbb{M}[i]$

Input: $(\texttt{write}, i, x) \in [1, n] \times \Sigma$
  if ACTIVE **and not** INTRUSION **then**
    $ctr \leftarrow ctr + 1$
    HIST $\leftarrow$ HIST $||$ $(\texttt{W}, i, x, ctr)$
    $\mathbb{M}[i] \leftarrow x$

Interface S_I

Input: $(\texttt{delete}, i) \in [1, n]$
  if INTRUSION **then**
    $\mathbb{M}[i] \leftarrow \epsilon$

Input: $(\texttt{restore}, i) \in [1, n]$
  if INTRUSION **then**
    if $\exists k, x$: HIST$[k] = (\texttt{W}, i, x, ctr)$ **then**
      $\mathbb{M}[i] \leftarrow x$

Fig. 3. The authentic SMR of [12] (only the differences with **SMR** are shown)

successful audit (returning `accept`), this guarantee holds until the server gains write-access to the storage, in which case a new audit has to reveal whether modifications have been made. We present the specification of the auditable and authentic SMR denoted by $\mathbf{aSMR}^{\mathsf{audit}}_{\Sigma,n}$ in fig. 4.

2.3 Expander graphs and expander codes

We recall the definitions and well known properties of expander graphs and expander codes. We follow the presentation of [16].

Let $G = (V, E)$ be an undirected d-regular graph on n vertices. The *expansion* of G is $\lambda := \max\{\lambda_2, |\lambda_n|\}$, where $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$ are the eigenvalues of the adjacency matrix of G. We say that G is a *Ramanujan* graph if $\lambda \leq 2\sqrt{d-1}$.

For a vertex $v \in V$, let $\Gamma(v)$ be the set of vertices adjacent to v. Let $\mathcal{C}_0 \subseteq \mathbb{F}_q^d$ be a linear code, called the *inner code*. Fix an order on the edges incident to each vertex of G, and let $\Gamma_i(v)$ be the i-th neighbor of v.

Using the graph G and the inner code $\mathcal{C}_0$ we can construct a new code, called an *expander code*. The expander code $\mathcal{C} := \mathcal{C}(G, \mathcal{C}_0)$ is defined as the set of all labelings of the edges of G that respect the inner code $\mathcal{C}_0$. It has length nd. More precisely, we have the following definition.

Definition 1 (Expander Code). *Let $\mathcal{C}_0 \subseteq \mathbb{F}_q^d$ be a linear code, and let $G = (V, E)$ be a d-regular expander graph on n vertices. The expander code $\mathcal{C}(G, \mathcal{C}_0) \subseteq$*

Resource $\mathbf{aSMR}^{\mathsf{audit}}_{\Sigma,n}$

Interface C

Input: $(\texttt{write}, i, x) \in [n] \times \Sigma$
Defined as in **aSMR** except the version number ctr has been removed.
Input: `audit`
if ACTIVE **and not** INTRUSION **then**
output `auditReq` to S_H
Let $d \in \{\text{allow}, \text{abort}\}$ be the result
if $d = \text{allow}$ **then**
$\mathbb{M}' \leftarrow [\,]$
for $i = 1$ to n **do**
if $\exists k, x$: HIST$[k] = (\texttt{W}, i, x)$ **then**
$k_0 \leftarrow \max\{k \mid \exists x$: HIST$[k] = (\texttt{W}, i, x)\}$
Parse HIST$[k_0]$ as $(\texttt{W}, i, x_0)$
$\mathbb{M}'[i] \leftarrow x_0$
else
$\mathbb{M}'[i] \leftarrow \lambda$
if $\mathbb{M}' = \mathbb{M}$ **then**
return `accept`
else
return `reject`
else
return `reject`

Interface S_I

Input: $(\texttt{restore}, i) \in [n]$
if INTRUSION **then**
if $\exists k, x$: HIST$[k] = (\texttt{W}, i, x)$ **then**
$k_0 \leftarrow \max\{k \mid \exists x$: HIST$[k] = (\texttt{W}, i, x)\}$
Parse HIST$[k_0]$ as $(\texttt{W}, i, x_0)$
$\mathbb{M}[i] \leftarrow x_0$
else
$\mathbb{M}[i] \leftarrow \lambda$

Fig. 4. Description of the auditable and authentic SMR of [1] (only the differences with the **aSMR** of fig. 3 are shown)

$\mathbb{F}_q^E$ is a linear code of length nd, so that for $c \in \mathbb{F}_q^E$, $c \in \mathcal{C}$ if and only if, for all $v \in V$,

$$\left(c_{(v,\Gamma_1(v))}, \ldots, c_{(v,\Gamma_d(v))}\right) \in \mathcal{C}_0$$

If $\mathcal{C}_0$ is a linear code of rate R_0, then $\mathcal{C}(G, \mathcal{C}_0)$ is a linear code of rate at least $2R_0 - 1$.

We say that an undirected graph $G = (L \cup R, E)$ is bipartite if, for all vertices $v \in L$, we have $\Gamma(v) \cap L = \emptyset$ and, for all vertices $v \in R$, we have $\Gamma(v) \cap R = \emptyset$. Let $G = (V, E)$ be an undirected d-regular graph on n vertices with expansion λ. From G, we can construct a d-regular graph $\tilde{G}$ on $2n$ vertices with expansion λ in the following way. The *double-cover* of G is the bipartite graph $\tilde{G} = (L \cup R, \tilde{E})$ defined as follows; let L and R be two copies of V. There is an edge between $u \in L$ and $v \in R$ if and only if $(u, v) \in E$.

It is known that expander codes constructed from bipartite graphs have good distance [18,21] :

Proposition 1. *Let $\mathcal{C}_0 \subseteq \mathbb{F}_q^d$ be a linear code with relative distance δ, and let $G = (L \cup R, E)$ be a d-regular bipartite expander graph with expansion λ. Then the expander code $\mathcal{C}(G, \mathcal{C}_0)$ has distance at least $\delta(\delta - \lambda/d)$.*

Moreover, $\mathcal{C}$ can be efficiently uniquely decoded up to this fraction of erasures [18,21]. The decoder and a proof of the following proposition can be found in app. B.

Proposition 2. *Let $\mathcal{C}_0 \subseteq \mathbb{F}_q^d$ be a linear code with relative distance δ. Let $\mathcal{D}(d)$ be the time needed to uniquely decode $\mathcal{C}_0$ from $\delta - 1/d$ erasures. Let $G = (L \cup R, E)$ be a d-regular bipartite expander graph on n vertices with expansion λ. Let $\epsilon > 0$ and suppose that $\frac{\lambda}{d} < \frac{\delta}{2}$. Then, there is an algorithm which uniquely decodes the expander code $\mathcal{C}(G, \mathcal{C}_0)$ from up to $(1 - \epsilon)\delta(\delta - \lambda/d)$ erasures in time $n \cdot \mathcal{D}(d)/\epsilon$.*

3 PoR with expander codes

3.1 Audit

Our scheme will use a generic audit for erasure codes presented in the CC framework by Badertscher and Maurer in [1]. We give the description and the security proof of [1].

First, we give notation for erasure codes. An (n, k, d) erasure code over the alphabet Σ with erasure symbol $\perp \notin \Sigma$, is a pair of algorithms $(\mathsf{enc}, \mathsf{dec})$ that satisfy : for all $F \in \Sigma^k$, let $\bar{F} := \mathsf{enc}(F) \in \Sigma^n$ and define the set

$$E := \{W \in (\Sigma \cup \{\perp\})^n \mid \forall i, W_i \in \{\bar{F}_i, \perp\} \text{ and at most } d-1 \text{ positions of } W \text{ are equal to } \perp\}$$

Then, for all $W \in E$, we have $\mathsf{dec}(W) = F$.

We now describe how Badertscher and Maurer implemented the ideas of [10,17] to construct an $\mathbf{aSMR}^{\mathsf{audit}}_{\Sigma^k,1}$ from an $\mathbf{aSMR}_{\Sigma,n}$. Let $(\mathsf{enc}, \mathsf{dec})$ be an

(n, k, d) erasure code and $F \in \Sigma^k$ be the client's file. We describe the PoR scheme $\mathsf{ecPor} := (\mathsf{ecInit}, \mathsf{ecAudit})$ for erasure codes.

On input `init` to ecInit, the converter sends `init` to $\mathbf{aSMR}_{\Sigma,n}$ and computes the encoding $\bar{F} := \mathsf{enc}(F) \in \Sigma^n$. Then, for each location $i \in [n]$, the converter sends $(\mathtt{write}, i, \bar{F}_i)$ to $\mathbf{aSMR}_{\Sigma,n}$.

On input $(\mathtt{read})$ to either ecInit or $\mathsf{ecAudit}$, the converter retrieves the whole memory content via $(\mathtt{read}, i)$ requests and obtains for each location, either a symbol $v_i \in \Sigma$ or the erasure symbol $\perp$. If v_i is returned, set $W_i := v_i$ else set $W_i :=\perp$. If $|\{i \in [n] \mid W_i =\perp\}| > d - 1$, the converter outputs ϵ at its outside interface, otherwise it computes $F := \mathsf{dec}(W)$, and outputs F.

Finally, on a query `audit` to converter $\mathsf{ecAudit}$, the converter chooses a random subset $S \subseteq [n]$ of size t and outputs $(\mathtt{read}, i)$ to $\mathbf{aSMR}$ for each $i \in S$ to retrieve the memory content at location i. If all read instructions for $i \in S$ returned a non-erased symbol, the converter outputs `accept`. Otherwise, it outputs `reject`. The integer t is chosen according to the security level we want to achieve. The security of the scheme is given by :

Theorem 1 ([1]). *Let $n, k, d \in \mathbb{N}$. Let $(\mathsf{enc}, \mathsf{dec})$ be an (n, k, d) erasure code for alphabet Σ and erasure symbol $\perp$. Let $\rho := 1 - \frac{d-1}{n}$ be the minimum fraction of symbols needed to recover the file. Then, the above protocol $\mathsf{ecPor} := (\mathsf{ecInit}, \mathsf{ecAudit})$ that chooses a random subset of size t during the audit, constructs the $\mathbf{aSMR}^{\mathtt{audit}}_{\Sigma^k,1}$ from the $\mathbf{aSMR}_{\Sigma,n}$ with respect to the simulator sim (described in the proof). More specifically, for all distinguishers $\mathbf{D}$ performing at most q audits,*

$$\Delta^{\mathbf{D}}(\mathsf{ecInit}_{\mathsf{C}_0}\mathsf{ecAudit}_{\mathsf{C}}\,\mathbf{aSMR}_{\Sigma,n}, \mathsf{sim}^{\mathsf{S}}\,\mathbf{aSMR}^{\mathtt{audit}}_{\Sigma^k,1}) \leq q \cdot \rho^t$$

Proof (Sketch [1]). We only compare the behaviors of the audit of the real system (the $\mathbf{aSMR}$ with the protocol) and of the ideal one (the $\mathbf{aSMR}^{\mathtt{audit}}$ with the simulator). The two systems behave in the same way in every other case, the reader can refer to [1] for a full proof.

Assume that a fraction α of cells of the real world $\mathbf{aSMR}$ have been deleted such that a $\beta := 1 - \alpha$ fraction is still available. A standard bound for binomial coefficients ensures that the probability of selecting a subset of memory locations containing no erased symbol during the audit is

$$\frac{\binom{\beta \cdot n}{|S|}}{\binom{n}{|S|}} \leq \beta^{|S|}$$

In the case when decoding would not be possible, *i.e.* if $\beta < \rho$, we see that the probability that the audit succeeds in the real world is no larger than $\rho^{|S|}$.

We describe the simulator sim. It maintains a simulated memory, emulating the real world memory, using the history of the ideal resource. On $(\mathtt{delete}, i)$, the simulator replaces the i-th entry of its simulated memory by ϵ. On $(\mathtt{restore}, i)$, the simulator restores the content of the i-th entry of its simulated memory to the last value written there. The simulator maintains a simulated history using the (ideal) history of the $\mathbf{aSMR}^{\mathsf{audit}}$.

If, after a `delete` request, the simulated memory contains d or more erased locations, the simulator deletes the whole ideal memory by sending `delete` requests to $\mathbf{aSMR}^{\mathsf{audit}}$. Similarly, if after a `restore` request, the simulated memory contains $d-1$ or less erased locations, the simulator restores the whole ideal memory by sending `restore` requests to $\mathbf{aSMR}^{\mathsf{audit}}$.

On an audit request, the simulator simulates the random locations probed during the audit by adding the appropriate `read` requests to its simulated history and evaluates if the audit succeeds by checking that none of the simulated locations are deleted. If so, it outputs `allow` to instruct the ideal resource to output the right result to the client, and otherwise it instructs the resource to output `reject`. The simulation is perfect until the following BAD event occurs : *having simulated a real audit, the simulator answers* `allow` *(audit should succeed) whereas d or more memory locations are currently erased.* It is the only case when simulation can differ from real execution. Thus, the probability of distinguishing can be upper bounded by the probability that event BAD happens in an execution. As discussed above, this happens with probability no larger than $q \cdot \rho^{|S|}$ over q audits. We give a detailed explanation of the BAD event in app. C.

3.2 Description of our PoR with expander codes : the general case

Let $\mathcal{C}_0$ be a linear code of length d, relative distance δ_0 and rate R_0. Using the Singleton bound, we have $\delta_0 \leq 1 + \frac{1}{d} - R_0$. Let G be a d-regular bipartite graph on n vertices with expansion λ.

We instantiate the PoR scheme $\mathsf{ecPor} := (\mathsf{ecInit}, \mathsf{ecAudit})$ with the expander code $\mathcal{C}(G, \mathcal{C}_0)$.

In the following, we determine the number t of edges probed during the audit needed to reach a given security level. If we suppose that $\frac{\lambda}{d} < \frac{\delta_0}{2}$, using the Singleton bound, we must have $R_0 < 1 + \frac{1}{d} - \frac{2\lambda}{d}$. Moreover, if $\mathcal{C}_0$ is Maximum Distance Separable, this implication becomes an equivalence. This is why, from now on, we will suppose that the inner code $\mathcal{C}_0$ is MDS.

We took G to be a bipartite expander graph with expansion λ such that $\frac{\lambda}{d} < \frac{\delta_0}{2}$. Using prop. 1, the minimum distance $\delta_{\mathcal{C}}$ of $\mathcal{C}(G, \mathcal{C}_0)$ is at least

$$\delta_0(\delta_0 - \frac{\lambda}{d}) > \frac{2\lambda^2}{d^2}$$

Let $\epsilon > 0$. If we want to correct a $(1-\epsilon)\delta_{\mathcal{C}}$ fraction of erasures, the minimum fraction of valid edges needed to recover our file is

$$\rho = 1 + \frac{1}{nd} - (1-\epsilon)\delta_{\mathcal{C}} \leq 1 + \frac{1}{nd} - (1-\epsilon)\frac{2\lambda^2}{d^2}$$

Let σ be a statistical security parameter and t be the number of edges probed during the audit. Our scheme is considered secure if $\rho^t \leq 2^{-\sigma}$. We want to choose t such that $t \geq -\sigma / \log \rho$.

Approximation : If $\frac{1}{nd} - (1-\epsilon)\frac{2\lambda^2}{d^2} \approx 0$, we have

$$\frac{-\sigma}{\log \rho} \approx \frac{nd^2\sigma}{2(1-\epsilon)n\lambda^2 - d} = \frac{d^2\sigma}{2(1-\epsilon)\lambda^2 - \frac{d}{n}}$$

Moreover, if G is Ramanujan, we have $\lambda \leq 2\sqrt{d-1}$ and

$$\frac{-\sigma}{\log \rho} \approx \frac{d\sigma}{8(1-\epsilon)}$$

If G has expansion $\sqrt{d}$ instead, we have

$$\frac{-\sigma}{\log \rho} \approx \frac{d\sigma}{2(1-\epsilon)}$$

Note that our scheme requires the adversary to only introduce erasures (and not errors). We enforce this using our authenticated server-memory resource (**aSMR**).

After a successful audit, the client can extract its file by running the decoder of prop. 2 which runs in time $\mathcal{O}(n \cdot \mathcal{D}(d)/\epsilon)$, where $\mathcal{D}(d)$ is the complexity of $\mathcal{C}_0$'s decoder.

3.3 Instantiation with the point-line incidence graph of the plane

Let Γ be the point-line incidence graph of the affine plane over $\mathbb{F}_q$ without the vertical lines. This graph is q-regular, has $2q^2$ vertices and expansion $\sqrt{q}$ (see the work of Tanner [20]). We have $\Gamma := (V_1 \cup V_2, E)$ where

$$V_1 := \{(x, y) \mid x, y \in \mathbb{F}_q\}, \quad V_2 := \{(a, b) \mid a, b \in \mathbb{F}_q\}$$

and

$$E := \{((x, y), (a, b)) \mid (x, y) \in V_1, (a, b) \in V_2, ax + b - y = 0\}$$

This graph is an excellent choice for our PoR scheme. Recall that the rate of the inner code is upper bounded by $1 + \frac{1}{d} - \frac{2\lambda}{d}$ and the rate of the expander code is lower bounded by $2R_0 - 1$. Moreover, going from expansion $2\sqrt{q-1}$ (the expansion of a q-regular Ramanujan graph) to $\sqrt{q}$ permits us to decrease the number of edges probed during the audit- and thus the communication complexity of our PoR -by a factor of 4. See sec. 3.4 for a comparison of the parameters of our PoR using the graph Γ and a Ramanujan graph.

The graph Γ also has a nice ratio between its regularity q and its number of edges q^3. Since we need to probe a number of edges linear in q, this ensures that our PoR scheme has communication complexity of order cubic root of the size of the outsourced file. This is in line or even better than other code-based PoR schemes, such as [11] (which has communication complexity of order square root

of the file size for $m = 2$). In sec. 3.4, we will show that, at a given security level, our PoR scheme stores much larger files than the lifted code-based PoR scheme of [11].

Our inner code $\mathcal{C}_0$ will be a Reed-Solomon code of rate $R_0 < 1 + \frac{1}{d} - \frac{2\lambda}{d}$. This code is MDS and thus, we can use the decoder of prop. 2 for our extraction phase. Moreover, because our inner code is a Reed-Solomon code, we can use the following result of Beelen *et al.* [2].

Let $\mathbb{F}_q := \{\alpha_1, \alpha_2, \ldots, \alpha_q\}$. We use the following labeling (of [2]) for the edges of Γ : if $(x, y) \in V_1$, $\Phi_{(x,y)}(i) := (x, y, \alpha_i, y - x\alpha_i)$ and, if $(a, b) \in V_2$, $\Phi_{(a,b)}(i) := (\alpha_i, a\alpha_i + b, a, b)$.

When q is a power of 2 or a prime, Beelen *et al.* [2] showed that when using this labeling on the graph Γ with a Reed-Solomon code of rate $1/2 < R_0 \leq 1$ as inner code, we obtain an expander code of rate *exactly* $R := R_0^3 + R_0(1 - R_0)(2R_0 - 1)$.

3.4 Parameters

Let σ be the statistical security parameter ($\sigma = 40$) and κ be the computational security parameter[3] ($\kappa = 128$). Set q, a power of 2. Let G be the q-regular point-line incidence graph on the affine plane $\mathbb{F}_q^2$. This graph has $2q^2$ vertices, q^3 edges and expansion $\lambda := \sqrt{q}$.

Let the inner code $\mathcal{C}_0$ be a Reed-Solomon code of length q and rate $R_0 = \max\{\frac{k}{q} \mid k \in \mathbb{N} \text{ and } \frac{k}{q} < 1 + \frac{1}{q} - \frac{2\lambda}{q}\}$. We take R_0 to be as big as possible (to reduce the storage overhead of the PoR) while still having a quasi-linear time decoder for the expander code. Indeed, since q is a power of 2, $\mathcal{C}_0$ can be erasure decoded in time $\mathcal{O}(q \log^2 q)$ thanks to the decoder of Tang and Lin [19].

Our expander code $\mathcal{C}(G, \mathcal{C}_0)$ has length q^3, rate $R := R_0^3 + R_0(1-R_0)(2R_0-1)$ and alphabet $\mathbb{F}_q$. Let $|F|$ be the size of the outsourced file in bits. It is such that $|F| = Rq^3 \log(q)$. Using prop. 2, we get an erasure decoder for $\mathcal{C}(G, \mathcal{C}_0)$ (and thus an extraction phase) running in time $\mathcal{O}(2q^3 \log^2 q)$ which is quasi-linear in the input size $q^3 \log q$. The storage overhead is given by $1/R - 1$, which is the redundancy of the code. The parameters of our PoR scheme and their asymptotic behavior are given in table 1. Even though our PoR has the same asymptotic behavior than the PoR of [11], we will show below that we get much better parameters in practice.

In table 2, we give concrete parameters of our PoR scheme for different values of q. Let us compare our scheme with the PoR of [11]. For $q = 512$, [11, Fig. 6] gives a code-based PoR with codewords of length q^3 and storage overhead of 68%. In table 2, we see that for $q = 512$ and codeword length q^3, our PoR has storage overhead of 21%, the same communication complexity as [11] and a quasi-linear time extraction phase. Moreover, the security analysis of [12] shows that the security of the PoR of [11] was overestimated, which means that, for a similar security level, our PoR compares even more favourably than in the previous example.

[3] of the MAC used to construct the **aSMR**

	Exact value	Asymptotics ($\|F\| \to \infty$)
C. storage overhead	κ	$\mathcal{O}(1)$
S. storage overhead	$(\frac{1}{R}-1)\|F\| + q^3\kappa$	$\mathcal{O}(\|F\|)$
comm. C. $\to$ S.	$\frac{q\sigma}{2}\log(q^3)$	$\mathcal{O}(\|F\|^{\frac{1}{3}}\log\|F\|)$
comm. S. $\to$ C.	$\frac{q\sigma}{2}(\kappa+\log q)$	$\mathcal{O}(\|F\|^{\frac{1}{3}}\log\|F\|)$

Table 1. The parameters of our scheme when using the point-line incidence graph over $\mathbb{F}_q^2$ and a Reed-Solomon code as inner code. $|F|$ denotes the file size in bits, κ the security parameter of the MAC, σ the statistical security parameter and R the rate of the code. We have $Rq^3\log(q) = |F|$.

q	R_0	R	$2q^2$	$\|F\|$	$\frac{1}{R}-1$	comm./$\|F\|$
256	0.878	0.758	131,072	12Mo	0.320	2×10^{-4}
512	0.913	0.827	524,288	124Mo	0.210	6×10^{-5}
1024	0.938	0.876	2,097,152	1.176Go	0.141	1×10^{-5}
2048	0.956	0.912	8,388,608	10.772Go	0.096	3×10^{-6}
4096	0.968	0.936	33,554,432	96.485Go	0.068	8×10^{-7}
8192	0.978	0.956	134,217,728	854.055Go	0.046	2×10^{-7}

Table 2. Effective parameters of our PoR using the point-line incidence graph over $\mathbb{F}_q^2$ for different values of q and Reed-Solomon codes as inner code. The graph is q-regular with $2q^2$ vertices. We choose the largest possible rate yielding a quasi-linear time decoder. The statistical security parameter is 40.

To give some perspective on the impact of the choice of the point-line incidence graph on the parameters of the PoR, we give the parameters of our scheme using a Ramanujan graph of the same size with expansion $2\sqrt{q-1}$ instead of $\sqrt{q}$. These parameters can be found in table 3.

q	R_0	R	$\|F\|$	$\frac{1}{R}-1$	comm./$\|F\|$
256	0.754	0.509	8Mo	0.965	1×10^{-4}
512	0.825	0.651	98Mo	0.537	2×10^{-5}
1024	0.876	0.752	1.009Go	0.330	4×10^{-6}
2048	0.912	0.824	9.735Go	0.213	1×10^{-6}
4096	0.938	0.875	90.246Go	0.142	2×10^{-7}
8192	0.956	0.912	814.614Go	0.097	5×10^{-8}

Table 3. Effective parameters of our PoR using a q-regular Ramanujan graph with $2q^2$ vertices and expansion $2\sqrt{q-1}$ for different values of q and a Reed-Solomon code as inner code. The graph is q-regular with $2q^2$ vertices. We choose the largest possible rate yielding a quasi-linear time decoder. The statistical security parameter is 40.

References

1. Badertscher, C., Maurer, U.: Composable and robust outsourced storage. In: Topics in Cryptology - CT-RSA 2018 - The Cryptographers' Track at the RSA Conference 2018, San Francisco, CA, USA, April 16-20, 2018, Proceedings. pp. 354–373 (2018). https://doi.org/10.1007/978-3-319-76953-0_19
2. Beelen, P., Høholdt, T., Piñero, F., Justesen, J.: On the dimension of graph codes with reed-solomon component codes. In: 2013 IEEE International Symposium on Information Theory. pp. 1227–1231 (2013). https://doi.org/10.1109/ISIT.2013.6620422
3. Bowers, K.D., Juels, A., Oprea, A.: Proofs of retrievability: Theory and implementation. In: Proceedings of the 2009 ACM Workshop on Cloud Computing Security. pp. 43–54. CCSW '09, ACM, New York, NY, USA (2009). https://doi.org/10.1145/1655008.1655015
4. Dodis, Y., Vadhan, S., Wichs, D.: Proofs of retrievability via hardness amplification. In: Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography. pp. 109–127. TCC '09, Springer-Verlag, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00457-5_8
5. Guo, A., Kopparty, S., Sudan, M.: New affine-invariant codes from lifting. In: Proceedings of the 4th Conference on Innovations in Theoretical Computer Science. pp. 529–540. ITCS '13, ACM, New York, NY, USA (2013). https://doi.org/10.1145/2422436.2422494
6. Høholdt, T., Justesen, J.: Graph codes with reed-solomon component codes. In: 2006 IEEE International Symposium on Information Theory. pp. 2022–2026 (2006). https://doi.org/10.1109/ISIT.2006.261904
7. Høholdt, T., Justesen, J.: The minimum distance of graph codes. In: Proceedings of the Third International Conference on Coding and Cryptology. p. 201âĂŞ212. IWCC'11, Springer-Verlag, Berlin, Heidelberg (2011)
8. Hoory, S., Linial, N., Wigderson, A.: Expander graphs and their applications. Bull. Amer. Math. Soc. **43**(04), 439âĂŞ562 (Aug 2006). https://doi.org/10.1090/s0273-0979-06-01126-8
9. Jost, D., Maurer, U.: Overcoming Impossibility Results in Composable Security Using Interval-Wise Guarantees, pp. 33–62. Advances in Cryptology - CRYPTO 2020, Springer International Publishing (2020)
10. Juels, A., Kaliski, Jr., B.S.: Pors: Proofs of retrievability for large files. In: Proceedings of the 14th ACM Conference on Computer and Communications Security. pp. 584–597. CCS '07, ACM, New York, NY, USA (2007). https://doi.org/10.1145/1315245.1315317
11. Lavauzelle, J., Levy-Dit-Vehel, F.: New proofs of retrievability using locally decodable codes. In: International Symposium on Information Theory ISIT 2016. pp. 1809 – 1813. Barcelona, Spain (2016). https://doi.org/10.1109/ISIT.2016.7541611
12. Levy-Dit-Vehel, F., Roméas, M.: A framework for the design of secure and efficient proofs of retrievability. Cryptology ePrint Archive, Report 2022/064 (2022), https://ia.cr/2022/064
13. Maurer, U.: Constructive Cryptography - A New Paradigm for Security Definitions and Proofs, pp. 33–56. Theory of Security and Applications, Springer Berlin Heidelberg (2012)
14. Maurer, U., Renner, R.: Abstract cryptography. In: In Innovations In Computer Science. Tsinghua University Press (2011)

15. Paterson, M., Stinson, D., Upadhyay, J.: A coding theory foundation for the analysis of general unconditionally secure proof-of-retrievability schemes for cloud storage. Journal of Mathematical Cryptology **7**(3), 183–216 (2013). https://doi.org/doi:10.1515/jmc-2013-5002, https://doi.org/10.1515/jmc-2013-5002
16. Ron-Zewi, N., Wootters, M., Zémor, G.: Linear-time erasure list-decoding of expander codes. In: 2020 IEEE International Symposium on Information Theory (ISIT). pp. 379–383 (2020). https://doi.org/10.1109/ISIT44484.2020.9174325
17. Shacham, H., Waters, B.: Compact proofs of retrievability. In: Pieprzyk, J. (ed.) Advances in Cryptology - ASIACRYPT 2008. pp. 90–107. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
18. Sipser, M., Spielman, D.: Expander codes. IEEE Transactions on Information Theory **42**(6), 1710–1722 (1996). https://doi.org/10.1109/18.556667
19. Tang, N., Lin, Y.: Fast encoding and decoding algorithms for arbitrary (n, k) Reed-Solomon codes over $\mathbb{F}_{2^m}$. IEEE Communications Letters **24**(4), 716–719 (2020). https://doi.org/10.1109/LCOMM.2020.2965453
20. Tanner, R.M.: Explicit concentrators from generalized n-gons. SIAM Journal on Algebraic Discrete Methods **5**(3), 287–293 (1984). https://doi.org/10.1137/0605030, https://doi.org/10.1137/0605030
21. Zémor, G.: On expander codes. IEEE Transactions on Information Theory **47**(2), 835–837 (2001), https://doi.org/10.1109/18.910593

A Background on Constructive Cryptography

A.1 Resources, Converters and Distinguishers

A *resource* $\mathbf{R}$ is a system that interacts, in a black-box manner, at one or more of its *interfaces*, by receiving an input at a given interface and subsequently sending an output at the same interface. Do note that a resource only defines the observable behavior of a system and not how it is defined internally. We use the notation $[\mathbf{R}_1, \ldots, \mathbf{R}_k]$ to denote the parallel composition of resources. It corresponds to a new resource and, if $\mathbf{R}_1, \ldots, \mathbf{R}_k$ have disjoint interfaces sets, the interface set of the composed resource is the union of those.

In CC, *converters* are used to link resources and reprogram interfaces, thus expressing the local computations of the parties involved. A converter is plugged on a set of interfaces at the inside and provides a set of interfaces at the outside. When it receives an input at its outside interface, the converter uses a bounded number of queries to the inside interface before computing a value and outputting it at its outside interface.

A converter π connected to the interface set $\mathcal{I}$ of a resource $\mathbf{R}$ yields a new resource $\mathbf{R}' := \pi^{\mathcal{I}}\mathbf{R}$. The interfaces of $\mathbf{R}'$ inside the set $\mathcal{I}$ are the interfaces emulated by π. A protocol can be modelled as a tuple of converters with pairwise disjoint interface sets.

A *distinguisher* $\mathbf{D}$ is an environment that connects to all interfaces of a resource $\mathbf{R}$ and sends queries to them. At any point, the distinguisher can end its interaction by outputting a bit. Its advantage is defined as $\Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{S}) := |\Pr[\mathbf{D}(\mathbf{R}) = 1] - \Pr[\mathbf{D}(\mathbf{S}) = 1]|$.

In this work, we make statements about resources with interface sets of the form $\mathcal{I} := \mathcal{P} \cup \{\mathsf{S}, \mathsf{W}\}$, where $\mathcal{P} := \{\mathsf{C}_0, \mathsf{C}\}$ is the set of honest client interfaces. A protocol is a tuple of converters $\pi := (\pi_{\mathsf{C}_0}, \pi_{\mathsf{C}})$. The goal of this protocol is to construct a so-called ideal resource from an available real resource in presence of a potentially dishonest server S. The world interface W models the direct influence of a distinguisher on a resource.

A.2 Specifications and Relaxations

An important concept of CC is the one of *specifications*. Systems are grouped according to desired or assumed properties that are relevant to the user, while other properties are ignored on purpose. A specification $\mathcal{S}$ is a set of resources that have the same interface set and share some properties, for example confidentiality. In order to construct this set of confidential resources, one can use a specification of assumed resources $\mathcal{R}$ and a protocol π, and show that the specification $\pi\mathcal{R}$ satisfies confidentiality. Proving security is thus proving that $\pi\mathcal{R} \subseteq \mathcal{S}$, sometimes written as $\mathcal{R} \xrightarrow{\pi} \mathcal{S}$, and we say that the protocol π constructs the specification $\mathcal{S}$ from the specification $\mathcal{R}$. The composition property of the framework comes from the transitivity of inclusion. Formally, for specifications $\mathcal{R}, \mathcal{S}$ and $\mathcal{T}$ and protocols π for $\mathcal{R}$ and π' for $\mathcal{S}$, we have $\mathcal{R} \xrightarrow{\pi} \mathcal{S} \wedge \mathcal{S} \xrightarrow{\pi'} \mathcal{T} \Rightarrow \mathcal{R} \xrightarrow{\pi' \circ \pi} \mathcal{T}$.

We use the real-world/ideal-world paradigm, and often refer to $\pi\mathcal{R}$ and $\mathcal{S}$ as the real and ideal-world specifications respectively, to understand security statements. Those statements say that the real-world is "just as good" as the ideal one, meaning that it does not matter whether parties interact with an arbitrary element of $\pi\mathcal{R}$ or one of $\mathcal{S}$. This means that the guarantees of the ideal specification $\mathcal{S}$ also apply in the real world where an assumed resource is used together with the protocol.

In this work, we use *simulators*, *i.e.*, converters that translate behaviors of the real world to the ideal world, to make the achieved security guarantees obvious. For example, one can model confidential servers as a specification $\mathcal{S}$ that only leaks the data length, combined with an arbitrary simulator σ, and show that $\pi\mathcal{R} \subseteq \sigma\mathcal{S}$. It is then clear that the adversary cannot learn anything more that the data length.

In order to talk about computational assumptions, post-compromise security or other security notions, the CC framework relies on *relaxations* which are mappings from specifications to larger, and thus weaker, *relaxed specifications*. The idea of relaxation is that, if we are happy with constructing specification $\mathcal{S}$ in some context, then we are also happy with constructing its relaxed variant. One common example of this is computational security. Let ϵ be a function that maps distinguishers $\mathbf{D}$ to the winning probability, in $[0, 1]$, of a modified distinguisher $\mathbf{D}'$ (the reduction) on the underlying computational problem. Formally,

Definition 2. *Let ϵ be a function that maps distinguishers to a value in $[0, 1]$. Then, for a resource $\mathbf{R}$, the reduction relaxation $\mathbf{R}^\epsilon$ is defined as $\mathbf{R}^\epsilon := \{\mathbf{S} \mid \forall \mathbf{D}, \Delta^{\mathbf{D}}(\mathbf{R}, \mathbf{S}) \leq \epsilon(\mathbf{D})\}$. This (in fact any) relaxation can be extended to a specification $\mathcal{R}$ by defining $\mathcal{R}^\epsilon := \cup_{\mathbf{R} \in \mathcal{R}} \mathbf{R}^\epsilon$.*

B Unique erasure decoding of expander codes

We present the unique erasure decoder for expander codes of prop. 2 that we recall below.

Proposition 2. *Let $\mathcal{C}_0 \subseteq \mathbb{F}_q^d$ be a linear code with relative distance δ. Let $\mathcal{D}(d)$ be the time needed to uniquely decode $\mathcal{C}_0$ from $\delta - 1/d$ erasures. Let $G = (L \cup R, E)$ be a d-regular bipartite expander graph on n vertices with expansion λ. Let $\epsilon > 0$ and suppose that $\frac{\lambda}{d} < \frac{\delta}{2}$. Then, there is an algorithm which uniquely decodes the expander code $\mathcal{C}(G, \mathcal{C}_0)$ from up to $(1-\epsilon)\delta(\delta - \lambda/d)$ erasures in time $n \cdot \mathcal{D}(d)/\epsilon$.*

We use the algorithm given in [16] and we follow their presentation and proof. This algorithm is well-known and follows from [18,21], we describe it in fig. 5. Since we use it in the extraction phase of our PoR, we prove its correction and complexity.

Algorithm UniqueDecode

Input: A description of the graph $G = (L \cup R, E)$ of degree d and of the code $\mathcal{C}_0 \subseteq \Sigma^d$ of relative minimum distance δ, and $z \in (\Sigma \cup \{\perp\})^E$.

Output: The unique $c \in \mathcal{C}(G, \mathcal{C}_0)$ so that c agrees with z on all un-erased positions.

$E_1 := \{e \in E \mid z_e \neq \perp\}$
$P_0 := \{v \in R \mid v \text{ is incident to an edge } e \in E \setminus E_1\}$
$P_1 := \{v \in L \mid v \text{ is incident to an edge } e \in E \setminus E_1\}$
for $t = 2, 3, \ldots$ **do**
    if $P_{t-1} = \emptyset$ **then**
        Return the fully labeled codeword.
    $P_t \leftarrow \emptyset$ and $E_t \leftarrow E_{t-1}$.
    for each vertex $v \in P_{t-1}$ so that $|(\{v\} \times \Gamma(v)) \cap E_{t-1}| > (1-\delta)d$ **do**
        Run $\mathcal{C}_0$'s erasure-correction algorithm to assign labels to the edges incident to v.
        Remove v from P_{t-1}.
        For any $(v, u) \notin E_{t-1}$, add (v, u) to E_t.
    For each vertex $v \in P_{t-1}$, for any $(v, u) \notin E_{t-1}$, add u to P_t.

Fig. 5. Unique expander code erasure decoder from up to $\delta(\delta - \lambda/d)(1-\epsilon)$ where λ is the expansion of the graph G and $\epsilon > 0$.

We will need to use the expander mixing lemma.

Theorem 2 (Expander Mixing Lemma, see *e.g.* [8]). *Suppose that $G = (L \cup R, E)$ is the double cover of a d-regular expander graph on n vertices with*

expansion λ. Then, for any $S \subseteq L$ and $T \subseteq R$,

$$\left| E(S,T) - \frac{d}{n}|S||T| \right| \leq \lambda\sqrt{|S||T|}$$

where $E(S,T)$ denotes the set of edges with endpoints in $S \cup T$.

First, notice that on any iteration $t \geq 2$, E_{t-1} is the subset of edges that have already been labeled before this iteration, and $P_{t-2} \cup P_{t-1}$ is the set of vertices touching an edge in $E \setminus E_{t-1}$ that is still erased. The following lemma bounds the size of P_t, and thus the number of steps the algorithm takes before terminating on line 6.

Lemma 1 ([16]). *The following holds:*

1. *For any $t \geq 1$, $|P_{t+1}| \leq (1-\epsilon)(\delta - \frac{\lambda}{d})n$*
2. *For any $t \geq 2$, $|P_{t+1}| \leq (\frac{1}{1+\epsilon})^2|P_t|$*

Proof. For any $t \geq 1$, let $B_{t-1} \subseteq P_{t-1}$ be the subset of vertices $v \in P_{t-1}$ that are incident to less than $(1-\delta)d$ un-erased edges in E_{t-1}. We have

$$P_{t+1} \subseteq B_{t-1} \subseteq P_{t-1} \tag{1}$$

since all vertices $v \in P_{t-1} \setminus B_{t-1}$ are removed from P_{t-1} on line 10, and consequently will not be in P_{t+1}.

For the first item, we have $|P_3| \leq |B_1|$ by (1), and that $|B_1| \leq (\delta - \lambda/d)(1-\epsilon)n$ since there are at most $(1-\epsilon)\delta(\delta-\lambda/d)nd$ erasures to begin with. Moreover, we have that $|P_3| \geq |P_5| \geq |P_7| \geq \ldots$, thus $|P_{t+1}| \leq (1-\epsilon)(\delta-\lambda/d)n$ for any even $t \geq 1$. Using a similar technique, one can show that this holds for any odd $t \geq 1$.

For the second item, the expander mixing lemma implies, for $t \geq 2$,

$$\delta d|B_{t-1}| \leq |E(B_{t-1}, P_t)| \leq \frac{d}{n}|B_{t-1}||P_t| + \lambda\sqrt{|B_{t-1}||P_t|}$$

as any vertex $v \in B_{t-1}$ has at least δd erased incident edges, and those edges are incident to P_t. After some rewriting, we have

$$|B_{t-1}| \leq \left(\frac{\lambda/d}{\delta - |P_t|/n}\right)^2 |P_t| \leq \left(\frac{1}{1+\epsilon}\right)^2 |P_t|$$

where the last inequality follows from the assumption that $\lambda/d \leq \delta/2$ and from $|P_t|/n \leq (1-\epsilon)(\delta - \lambda/d)$ by the first item. Using (1) we finally get

$$|P_{t+1}| \leq |B_{t-1}| \leq \left(\frac{1}{1+\epsilon}\right)^2 |P_t|$$

Using the above lemma, we conclude that after $\mathcal{O}((\log n)/\epsilon)$ iterations, P_{t-1} is empty and the algorithm terminates. Moreover, the amount of work done is at most

$$\mathcal{D}(d) \cdot \sum_{t=1}^{\infty} |P_t| = \mathcal{D}(d) \cdot n \sum_{t=1}^{\infty} \left(\frac{1}{1+\epsilon}\right)^{2t} = \mathcal{D}(d) \cdot \frac{n}{\epsilon},$$

where $\mathcal{D}$ is the complexity of $\mathcal{C}_0$'s erasure decoder, which proves the proposition.

C Detailed description of the **BAD** event

We here detail the interactions between the audit requests, the simulator, the distinguisher and the ideal resource, during an audit. It is very important to notice that the simulator is only connected to the interfaces S_I and S_H of the server, and has no interaction with the clients. As `audit` is a functionality available at clients' interfaces, it follows that an audit request is not directly treated by the simulator. As described in fig.4, the following steps are done when the ideal resource $\mathbf{aSMR}^{\mathsf{audit}}$ receives an `audit` request at the client interface C :

1. the resource sends `auditReq` at interface S_H.
2. via its interface S_H, the simulator either responds `allow` or `abort`.
3. if `allow`, the audit is run and the resource sends back to the client either `accept` or `reject`, depending on whether the ideal audit has succeeded or failed.
4. if `abort`, the ideal resource always sends `reject` back to the client.

Thus, in the proof, the simulator cannot directly control the outcome of the audit, in the sense that it cannot decide whether the ideal resource would send `accept` or `reject` back to the client. On the other hand, as it receives the `auditReq` request at interface S_H, the simulator can choose to answer `abort` or `allow` to the ideal resource. Let us now examine how the simulator behaves in both cases :

1. The simulator answers `abort` :
   It has received an audit request via `auditReq`. It then runs a simulation of a real audit (the one of the protocol) on its simulated memory: it chooses a set of t addresses, adds the corresponding `read` requests to its simulated history, and tests whether the values stored at those addresses (in its simulated memory) are not erased ($\neq \epsilon$). In this case, this test fails : at least one value is erased, and the simulator is convinced that the real audit has to fail. It thus decides to answer the ideal resource `abort`. Consequently, the ideal resource does not run an ideal audit, but rather answers the client `reject`. The client is in fact controlled by the distinguisher in the ideal setting. Looking at the simulated history, it (the distinguisher) believes to interact with the real resource.
2. The simulator answers `allow`:
   It has received an audit request via `auditReq`. It then runs a simulation of a real audit (the one of the protocol) on its simulated memory: it chooses a set of t addresses, adds the corresponding `read` requests to its simulated history, and tests whether the values stored at those addresses (in its simulated memory) are not erased ($\neq \epsilon$). In this case, this test succeeds, all values are valid, and the simulator is convinced of the success of the real audit, as it simulated it with success. It thus sends `allow` to the ideal resource. The subtlety lies here : the ideal resource receives `allow` but it does not imply that it will send `accept` back to the client. It will run its ideal audit, and send the outcome of this audit to the client. The point is, when the

simulator sends its answer (here `allow`) to the ideal resource, it already has simulated the real audit (here with success), and written the corresponding entries in its simulated history. Being not connected to the client's interface, the simulator does not "see" the result of the ideal audit (`accept` or `reject`) sent by the ideal resource to the client. Thus it cannot *a posteriori* modify its audit simulation to comply with the response of the ideal resource. The distinguisher, being connected to server and client's interfaces, has access to the ideal audit response and, by comparing it to the simulated history, can see the incoherence; (the trace of the audit in the simulated history corresponds to one which should fail). This incoherence never happens in the real resource, thus the distinguisher has distinguished between the two systems.